Nested Interval Behavior

Hierarchical Data Management in Relational Databases

using Nested Interval Node Key Encoding

for the Yii PHP Framework

Developed for the Yii Community by

PBM Web Development

PBM

Nested Interval Behavior

Contents

Nested Interval Behavior

Introduction

Nested Intervals provide a means of managing hierarchical data in a relational database by keying nodes in the tree using rational numbers.

The Nested Interval Behavior is an implementation of the encoding scheme presented in "Using Rational Numbers to Key Nested Sets"; Dan Hazel 19 June 2008. (http://arxiv.org/PS_cache/arxiv/pdf/0806/0806.3115v1.pdf)

The key of a node is calculated from the key of the parent node in such a way that places every descendant of a node before the next sibling of the node.

A key feature of the encoding is a significantly reduced requirement for re-indexing of nodes; addition or deletion of a last child node requires no re-indexing, and deletion of any other node only requires re-indexing of sibling sub-trees.

Nested Interval encoding also provides inherent support for multiple trees.

License

Nested Interval Behavior

Compatibility

	Yii	OS
Tested with	1.1.8	Windows 7
Should work with	1.1.x	All

Installation

1. Download Nested Interval Behavior from http://www.yiiframework.com/extension/nested-interval/
2. Open the zip file and extract the "nestedInterval" directory to the required place, e.g. under application.extensions.

Usage

Attach the Nested Interval Behavior to a model in the usual way. The figure below attaches the behavior with no additional configuration; see the API documentation for properties that can be configured.

```
class MyModel extends CActiveRecord {
        …
  public function behaviors() {
    return array(
      'nestedInterval'=>'path.to.NestedIntervalBehavior'
    );
  }
        …
}
```

Adding a node

When adding new data it must be added either as a child or sibling of an existing node, or as a root node; *CActiveRecord::save()* and *CActiveRecord::insert()* will throw exceptions if used to add data.

The following methods can be used to add data:

addAsChild()	Adds the target (new) node as a child of the owner node
addAsChildOf()	Adds the current (new) node a s a child of the target
addAsRoot()	Adds the current (new) node a s a new root node
addAsSibling()	Adds the target (new) node as a sibling of the owner node
addAsSiblingOf()	Adds the current (new) node a s a sibling of the target
after()	Adds the target (new) node as the next sibling of the owner node
append()	Adds the target (new) node as the last child of the owner node

appendTo()	Adds the current (new) node as the last child of the target node
before()	Adds the target (new) node as the previous sibling of the owner node
first()	Adds the target (new) node as the first sibling of the owner node
insertAfter()	Adds the current (new) node as the next sibling of the target node
insertBefore()	Adds the current (new) node as the previous sibling of the target node
insertFirst()	Adds the current (new) node as the first sibling of the target node
insertLast()	Adds the current (new) node as the last sibling of the target node
last()	Adds the target (new) node as the last sibling of the owner node
prepend()	Adds the target (new) node as the first child of the owner node
prependTo()	Adds the current (new) node as the first child of the target node

Deleting Nodes

Deletion of data must be done using one of the methods below; using *CActiveRecord::delete()* will throw an exception.

The following methods can be used to delete data:

deleteDescendants()	Deletes the descendants of the owner node
deleteNode()	Deletes the owner node and its descendants

Retrieving Data

Data can be retrieved in a variety of ways.

Scopes

Nested Interval Behavior provides a number of scopes that can be used to retrieve data:

ancestors()	Finds the ancestors of the owner node
child()	Finds a specific child of the owner node
children()	Finds the children of the owner node
descendants()	Finds the descendants of the owner node
firstChild()	Finds the first child of the owner node
firstSibling()	Finds the first sibling of the owner node

lastChild()	Finds the last child of the owner node
lastSibling()	Finds the last sibling of the owner node
nextSibling()	Finds the next sibling of the owner node
parent()	Finds the parent of the owner node
previousSibling()	Finds a specific sibling of the owner node
root()	Finds a root node. If an index (1 based) is given, that root node is found, otherwise the root node of the owner node is found.
roots()	Finds all root nodes
sibling()	Finds a specific sibling of the owner node
siblings()	Finds the siblings of the owner node

Getters

Each scope has an associated getter; these are wrappers for *$model->scope()->find[All]()* , e.g. *$model->getFirstChild()* to retrieve the first child of a node.

Properties

The getters also mean that most data can be accessed as properties of the owner node, e.g. *$model->firstChild* also retrieves the first child of a node. The exceptions are *NestedIntervalBehavior::getChild()* , *NestedIntervalBehavior::getRoot()* , and *NestedIntervalBehavior::getSibling()* as these require a number; accessing these as properties will return a node using the method's default value.

The following are equivalent and return the first child of a node; which to use is purely a matter of preference:

scope	$model->firstChild()->find();
getter	$model->getFirstChild();
property	$model->firstChild;

createDescendantTree() and createAncestorPath()

These methods make the children and the parent of a node available as related records of the node.

For example, *$model->createDescendantTree()* makes all the node's children available as related records, each child will have its children as related records, and so on. Similarly, *$model->createAncestorPath ()* makes the node's parent available as a related record, the parent will have its parent as a related record, and so on the root.

Node Information

There are a number of methods to obtain information about a node:

getAncestorCount()	Returns the number of ancestors of the owner node
getChildCount()	Returns the number of children of the owner node
getDescendantCount()	Returns the number of descendants of the owner node
getIsDeleted()	Returns a value indicating if the owner node has been deleted
getIsLeaf()	Returns a value indicating if the owner node is a leaf node, i.e. has no children
getIsRoot()	Returns a value indicating if the owner node is a root node
getLevel()	Returns the level of the owner node
getPosition()	Returns the position – sibling number – of the owner node
hasChildren()	Returns a value indicating if the owner node has children
isAncestorOf()	Returns a value indicating if the owner node is an ancestor of a node
isChildOf()	Returns a value indicating if the owner node is a child of a node
isDescendantOf()	Returns a value indicating if the owner node is a descendant of a node
isParentOf()	Returns a value indicating if the owner node is the parent of a node
isSiblingOf()	Returns a value indicating if the owner node is a sibling of a node

getXXX() methods allow the value to be accessed as a property, e.g. *$model->level* also returns the level of the node within the hierarchy.

Moving Nodes

There are a number of methods to move nodes and their descendants – moving a node always moves its descendants. Nodes can be moved within a tree or to another tree, and non-root nodes can be moved to become a root node. All move methods return the number of nodes moved.

Methods for moving nodes and their descendants are:

moveAfter()	Moves the owner node to become the next sibling of the target node
moveBefore()	Moves the owner node to become the previous sibling of the target node
moveToFirstChild()	Moves the owner node to become the first child of the target node
moveToFirstSibling()	Moves the owner node to become the first sibling of the target node

moveToLastChild()	Moves the owner node to become the last child of the target node
moveToLastSibling()	Moves the owner node to become the last sibling of the target node
moveToNthChild()	Moves the owner node to become the nth child of the target node
moveToNthSibling()	Moves the owner node to become the nth sibling of the target node
moveToRoot()	Moves the owner node to become a new root node

After being moved a node will be keyed to its new position. Therefore, for example, *$node->moveAfter($target); $prev = $node->previousSibling;* will result in *$target===$prev*

NestedIntervalBehavior

Inheritance	NestedIntervalBehavior » CActiveRecordBehavior » CBehavior » CComponent
Implements	IBehavior

Public Properties

See CActiveRecordBehavior for inherited properties.

Name	Type	Description
childRelatedRecords	string	The name of child related records. Defaults to "childNodes". Only used by createDescendantTree().
dv	string	The name of the attribute used to store the Denominator of the node key. Defaults to "dv"
nv	string	The name of the attribute used to store the Numerator of the node key. Defaults to "nv"
parentRelatedRecord	string	The name of the parent related record. Defaults to "parentNode". Only used by createAncestorPath().
sdv	string	The name of the attribute used to store the Next Sibling Denominator of the node key. Defaults to "sdv"
snv	string	The name of the attribute used to store the Next Sibling Numerator of the node key. Defaults to "snv"

Public Methods

See CActiveRecordBehavior for inherited methods.

Name	Description

Nested Interval Behavior

Name	Description
addAsChild()	Adds the target (new) node as a child of the owner node
addAsChildOf()	Adds the current (new) node a s a child of the target
addAsRoot()	Adds the current (new) node a s a new root node
addAsSibling()	Adds the target (new) node as a sibling of the owner node
addAsSiblingOf()	Adds the current (new) node a s a sibling of the target
after()	Adds the target (new) node as the next sibling of the owner node
ancestors()	Named scope to find the ancestors of the owner node
append()	Adds the target (new) node as the last child of the owner node
appendTo()	Adds the current (new) node as the last child of the target node
before()	Adds the target (new) node as the previous sibling of the owner node
child()	Named scope to find a specific child of the owner node
children()	Named scope to find the children of the owner node
createAncestorPath()	Makes the parent of the owner node available as a related record of the node, the parent's parent as a related record of the parent, and so on.
createDescendantTree()	Makes the children of the owner node available as related records of the node, each of the children's children as related records of the children and so on.
deleteDescendants()	Deletes the descendants of the owner node
deleteNode()	Deletes the owner node and its descendants
descendants()	Named scope to find the descendants of the owner node
first()	Adds the target (new) node as the first sibling of the owner node
firstChild()	Named scope to find the first child of the owner node
firstSibling()	Named scope to find the first sibling of the owner node
getAncestors()	Returns the ancestors of the owner node
getAncestorCount()	Returns the number of ancestors of the owner node
getChild()	Returns a child of the owner node
getChildCount()	Returns the number of children of the owner node

Nested Interval Behavior

Name	Description
getChildren()	Returns the children of the owner node
getDescendants()	Returns the descendants of the owner node
getDescendantCount()	Returns the number of descendants of the owner node
getFirstChild()	Returns the first child of the owner node
getFirstSibling()	Returns the first sibling of the owner node
getIsDeleted()	Returns a value indicating if the owner node has been deleted
getIsLeaf()	Returns a value indicating if the owner node is a leaf node, i.e. has no children
getIsRoot()	Returns a value indicating if the owner node is a root node
getLastChild()	Returns the last child of the owner node
getLastSibling()	Returns the last sibling of the owner node
getLevel()	Returns the level of the owner node
getNextSibling()	Returns the next sibling of the owner node
getParent()	Returns the parent of the owner node
getPosition()	Returns the position – sibling number – of the owner node
getPreviousSibling()	Returns the previous sibling of the owner node
getRoot()	Returns a specific root node. If an index (1 based) is given, that root node is returned, else the root node of the owner node is returned.
getSibling()	Returns a sibling of the owner node
getSiblingCount()	Returns the number of siblings of the owner node
getSiblings()	Returns the siblings of the owner node
hasChildren()	Returns a value indicating if the owner node has children
isAncestorOf()	Returns a value indicating if the owner node is an ancestor of a node
isChildOf()	Returns a value indicating if the owner node is a child of a node
isDescendantOf()	Returns a value indicating if the owner node is a descendant of a node
isLeaf()	Returns a value indicating if the owner node is a leaf node, i.e. has no children

Nested Interval Behavior

Name	Description
isRoot()	Returns a value indicating if the owner node is a root node
isParentOf()	Returns a value indicating if the owner node is the parent of a node
isSiblingOf()	Returns a value indicating if the owner node is a sibling of a node
insertAfter()	Adds the current (new) node as the next sibling of the target node
insertBefore()	Adds the current (new) node as the previous sibling of the target node
insertFirst()	Adds the current (new) node as the first sibling of the target node
insertLast()	Adds the current (new) node as the last sibling of the target node
last()	Adds the target (new) node as the last sibling of the owner node
lastChild()	Named scope to find the last child of the owner node
lastSibling()	Named scope to find the last sibling of the owner node
moveAfter()	Moves the owner node to become the next sibling of the target node
moveBefore()	Moves the owner node to become the previous sibling of the target node
moveToFirstChild()	Moves the owner node to become the first child of the target node
moveToFirstSibling()	Moves the owner node to become the first sibling of the target node
moveToLastChild()	Moves the owner node to become the last child of the target node
moveToLastSibling()	Moves the owner node to become the last sibling of the target node
moveToNthChild()	Moves the owner node to become the nth child of the target node
moveToNthSibling()	Moves the owner node to become the nth sibling of the target node
moveToRoot()	Moves the owner node to become a new root node
nextSibling()	Named scope to find the next sibling of the owner node
parent()	Named scope to find the parent of the owner node
prepend()	Adds the target (new) node as the first child of the owner node
prependTo()	Adds the current (new) node as the first child of the target node
previousSibling()	Named scope to find a specific sibling of the owner node
root()	Finds a root node. If an index (1 based) is given, that root node is found, else the root node of the owner node is found.

Nested Interval Behavior

Name	Description
roots()	Finds all root nodes
self()	Named scope to find the owner node. Used internally
sibling()	Named scope to find a specific sibling of the owner node
siblings()	Named scope to find the siblings of the owner node

Events

Name	Description
beforeDelete ()	Called before a node is deleted. Ensures that the node is being deleted by the behaviour.
beforeSave ()	Called before a node is saved. Ensures that a new node is being saved by the behaviour.

Property Details

childRelatedRecords

The name of child related records. Defaults to "childNodes". Only used by createDescendantTree().

dv

The name of the attribute used to store the Denominator of the node key. Defaults to "dv"

nv

The name of the attribute used to store the Numerator of the node key. Defaults to "nv"

parentRelatedRecords

The name of the parent related record. Defaults to "parentNode". Only used by createAncestorPath().

sdv

The name of the attribute used to store the Next Sibling Denominator of the node key. Defaults to "sdv"

snv

The name of the attribute used to store the Next Sibling Numerator of the node key. Defaults to "snv"

Method Details

addAsChild ($*target*, $*n* = 0, $*validate* = true, $*attributes* = null)

Adds the target node as a child of the owner node.

Parameters:

CActiveRecord	*Parent node*
integer	*Position to add at: 1 = 1st child, 2 = 2nd child, n = nth child; 0 (default) = last child - this provides the best performance as it minimises reindexing*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was appended, FALSE if not

addAsChildOf ($*target*, $*n* = 0, $*validate* = true, $*attributes* = null)

Adds the owner node as a child of the target node.

Parameters:

CActiveRecord	*Target node*
integer	*Position to add at: 1 = 1st child, 2 = 2nd child, n = nth child; 0 (default) = last child - this provides the best performance as it minimises reindexing*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was appended, FALSE if not

addAsRoot ($*validate* = true, $*attributes* = null)

Adds the owner node as a root node.

Parameters:

boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was added, FALSE if not

addAsSibling ($*target*, $*n* = 0, $*validate* = true, $*attributes* = null)

Adds the target node as the nth sibling of the owner node.

Parameters:

CActiveRecord	*Target node*

integer	*Position to add at: 1 = 1st sibling, 2 = 2nd sibling, n = nth sibling; 0 (default) = last sibling - this provides the best performance as it minimises reindexing*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

addAsSiblingOf ($*target*, $*n* = 0, $*validate* = true, $*attributes* = null)

Adds the owner node as the nth sibling of the target node.

Parameters:

CActiveRecord	Target node
integer	Position to add at: 1 = 1st sibling, 2 = 2nd sibling, n = nth sibling; 0 (default) = last sibling - this provides the best performance as it minimises reindexing
boolean	Whether to validate attributes
array	Attributes to save

Returns:

boolean TRUE if the node was inserted, FALSE if not

after ($*target*, $*validate* = true, $*attributes* = null)

Adds the target node as the next sibling of the owner node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

See also:

insertAfter()

ancestors ()

Named scope to find the ancestors of the owner. The default order is from the parent up through the tree.

Returns:

CActiveRecord The owner

append ($target, $validate = true, $attributes = null)

Appends the target node to the owner node; the target node becomes the last child of the owner node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was appended, FALSE if not

appendTo ($target, $validate = true, $attributes = null)

Appends the owner node to the target node; the owner node becomes the last child of the target node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was appended, FALSE if not

before ($target, $validate = true, $attributes = null)

Adds the target node as the previous sibling of the owner node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

See also:

insertBefore()

beforeDelete ($event)

Handle the owner's 'beforeDelete' event.

Parameters:

CEvent	*The event*

Returns:

boolean TRUE if the operation should continue, FALSE if not

Exceptions:

NestedIntervalException	*If CActiveRecord::delete() called directly on a node*

beforeSave ($*event*)

Handle the owner's 'beforeSave' event.

Parameters:

CEvent	*The event.*

Returns:

boolean TRUE if the operation should continue, FALSE if not

Exceptions:

NestedIntervalException	*If CActiveRecord::save() called directly on a new node*

child ($*n* = 1)

Named scope to find the nth child of a node.

Parameters:

integer	*The child number to return; $n==0 returns the last child. NULL will be returned if $n>lastChild*

Returns:

CActiveRecord The owner

children ()

Named scope to find the children of a node. The dataset is ordered in node key ascending order by default

Returns:

CActiveRecord The owner

createAncestorPath ()

Creates a path of ancestors under the owner node. After calling this method the owner will have its parent as related record, the parent will have its parent as related record, and so on.

Returns:

void

createDescendantTree ()

Creates a tree of descendants under the owner node. After calling this method the owner will have its children as related records, each of the children will have their children as related records, and so on.

Returns:

void

deleteDescendants ()

Delete the descendants of a node

Returns:

integer the number of nodes deleted.

deleteNode ()

Delete a node and all its descendants

Returns:

integer the number of nodes deleted.

descendants ()

Named scope to find the descendants of the owner. The dataset is ordered in node key ascending order unless overridden.

Returns:

CActiveRecord The owner

first ($*target*, $*validate* = true, $*attributes* = null)

Adds the target node as the first sibling of the owner node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Aattributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

See also:

insertFirst()

firstChild ()

Named scope to find the first child of a node.

Returns:

CActiveRecord The owner

firstSibling ()

Named scope to find the first child of the parent of the owner node.

Returns:

CActiveRecord The owner

getAncestorCount ()

Returns the number of ancestors of the owner

Returns:

integer The number of ancestors.

getAncestors ()

Returns the ancestors of the owner

Returns:

array The ancestors of the owner.

getChild ($*n*)

Returns the nth child of the owner

Returns:

CActiveRecord The nth child of the owner. NULL if it does not exist.

getChildCount ()

Returns the number of children the owner has.

Returns:

integer The number of children of the owner.

getChildren ()

Returns the children of the owner

Returns:

array Children of the owner. An empty array is returned if the owner has no children, i.e. is a leaf node.

getDescendantCount ()

Returns the number of descendants the owner has.

Returns:

integer The number of descendants of the owner.

getDescendants ()

Returns the descendants of the owner.

Returns:

array The descendants of the owner.

getFirstChild ()

Returns the first child of the owner.

Returns:

CActiveRecord The first child of the owner; NULL if the owner has no children

getFirstSibling ()

Returns the first sibling of the owner.

Returns:

CActiveRecord The first sibling of the owner

getIsDeleted ()

Returns a value indicating if this node has been deleted.

Returns:

boolean TRUE if this node has been deleted, FALSE if not

getIsLeaf ()

Returns a value indicating if the node is a leaf node, i.e. it has no children.

Returns:

boolean TRUE if the node is a leaf node; FALSE if not.

getIsRoot ()

Returns a value indicating if the node is a root node.

Returns:

boolean TRUE if the node is a root node; FALSE if not.

getLastChild ()

Returns the last child of the owner.

Returns:

CActiveRecord The last child of the owner; NULL if the owner has no children

getLastSibling ()

Returns the last sibling of the owner.

Returns:

CActiveRecord The last sibling of the owner

getLevel ()

Returns the level of the owner.

Returns:

integer The level of the owner.

getNextSibling ()

Returns the next sibling of the owner.

Returns:

CActiveRecord The next sibling of the owner; NULL if the owner is the last sibling

getParent ()

Returns the parent of the owner

Returns:

array The parent of the owner. Returns NULL is the owner is a root item.

getPosition ()

Returns the position of the owner relative to its siblings; i.e. 1=the 1st child/root, 2=the 2nd child/root, ..., n=nth child/root

Returns:

integer The position of the owner relative to its siblings

getPreviousSibling ()

Returns the previous sibling of the owner.

Returns:

CActiveRecord The previous sibling of the owner; NULL if the owner is the first sibling

getRoot ($*n* = 0)

Returns the nth root node. If $n==0 the root node of the owner is returned.

Returns:

CActiveRecord The nth root, or root of the owner.

getSibling ($*n*)

Returns the nth sibling of the owner

Returns:

CActiveRecord The nth sibling of the owner. NULL if it does not exist.

getSiblingCount ($*which* = self::SIBLINGS_EX)

Returns the number of siblings the owner has

Parameters:

integer	*Which siblings to count: + self::SIBLINGS_EX - all siblings excluding the owner (default) + self::SIBLINGS_ALL - all siblings including the owner - equivalent to the parent's children + self::SIBLINGS_AFTER - later siblings + self::SIBLINGS_BEFORE - earlier siblings*

Returns:

integer The number of siblings of the owner.

getSiblings ($*which* = self::SIBLINGS_EX)

Returns the siblings of the owner

Parameters:

integer	*Which siblings to return: + self::SIBLINGS_EX - all siblings excluding the owner (default) + self::SIBLINGS_ALL - all siblings including the owner - equivalent to the parent's children + self::SIBLINGS_AFTER - later siblings + self::SIBLINGS_BEFORE - earlier siblings*

Returns:

array The siblings of the owner.

hasChildren ()

Returns a value indicating if the node has children

Returns:

boolean TRUE if the node has children; FALSE if not.

insertAfter ($*target*, $*validate* = true, $*attributes* = null)

Adds the owner node as the next sibling of the target node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Aattributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

insertBefore ($*target*, $*validate* = true, $*attributes* = null)

Adds the owner node as the previous sibling of the target node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

insertFirst ($*target*, $*validate* = true, $*attributes* = null)

Adds the owner node as the first sibling of the target node

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

insertLast ($*target*, $*validate* = true, $*attributes* = null)

Adds the owner node as the last sibling of the target node

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

isAncestorOf ($*node*)

Returns a value indicating if the owner is an ancestor of the node

Returns:

boolean TRUE if the owner is an ancestor of the node; FALSE if not.

isChildOf ($*node*)

Returns a value indicating if the owner is a child of the node

Returns:

boolean TRUE if the owner is a chils of the node; FALSE if not.

isDescendantOf ($*node*)

Returns a value indicating if the owner is a descendant of the node

Returns:

boolean TRUE if the owner is a descendant of the node; FALSE if not.

isLeaf ()

Returns a value indicating if the node is a leaf node, i.e. it has no children.

Returns:

boolean TRUE if the node is a leaf node; FALSE if not.

isParentOf ($*node*)

Returns a value indicating if the owner is the parent of the node

Returns:

boolean TRUE if the owner is a descendant of the node; FALSE if not.

isRoot ()

Returns a value indicating if the node is a root node

Returns:

boolean TRUE if the node is a root node; FALSE if not.

isSiblingOf ($*node*)

Returns a value indicating if the owner is a descendant of the node

Returns:

boolean TRUE if the owner is a descendant of the node; FALSE if not.

last ($*target*, $*validate* = true, $*attributes* = null)

Adds the target node as the last sibling of the owner node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*

array	*Aattributes to save*

Returns:

boolean TRUE if the node was inserted, FALSE if not

See also:

insertLast()

lastChild ()

Named scope to find the last child of a node.

Returns:

CActiveRecord The owner

lastSibling ()

Named scope to find the last child of the parent of the owner node.

Returns:

CActiveRecord The owner

moveAfter ($*target*)

Moves the owner node to become the next sibling of the target node

Parameters:

CActiveRecord	*The target node*

Returns:

integer The number of nodes moved

moveBefore ($*target*)

Moves the owner node to become the previous sibling of the target node

Parameters:

CActiveRecord	*The target node*

Returns:

integer The number of nodes moved

moveToFirstChild ($*target*)

Moves the owner node to become the first child of the target node

Parameters:

CActiveRecord	*The target node*

Returns:

integer The number of nodes moved

moveToFirstSibling ($*target*)

Moves the owner node to become the first sibling of the target node

Parameters:

CActiveRecord	*The target node*

Returns:

integer The number of nodes moved

moveToLastChild ($*target*)

Moves the owner node to become the last child of the target node

Parameters:

CActiveRecord	*The target node*

Returns:

integer The number of nodes moved

moveToLastSibling ($*target*)

Moves the owner node to become the last sibling of the target node

Parameters:

CActiveRecord	*The target node*

Returns:

integer The number of nodes moved

moveToNthChild ($*target*, $*n*)

Moves the owner node to become the nth child of the target node

Parameters:

CActiveRecord	*The target node*
integer	*Position to move to: 1 = 1st child, 2 = 2nd child, n = nth child; 0 (default) = last child - this provides the best performance as it minimises re-indexing*

Returns:

integer The number of nodes moved

moveToNthSibling ($*target*, $*n*)

Moves the owner node to become the nth sibling of the target node

Parameters:

CActiveRecord	*The target node*

Returns:

integer The number of nodes moved

moveToRoot ()

Moves the owner node to become a new root node

Returns:

integer The number of nodes moved

nextSibling ()

Named scope to find the next sibling of the owner

Returns:

CActiveRecord The owner

parent ()

Named scope to find the parent of the owner

Returns:

CActiveRecord The owner

parentQuadruple ($*node* = null)

Returns an array with the node's quadruple

Parameters:

CActiveRecord	*The node to return the parent quadruple of. If NULL the owner's parent quadruple is returned*

Returns:

array The parent quadruple array(nv,dv,snv,sdv)

prepend ($*target*, $*validate* = true, $*attributes* = null)

Prepends the target node to the owner node, i.e. the target node becomes the first child of the owner node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was prepended, FALSE if not

prependTo ($*target*, $*validate* = true, $*attributes* = null)

Prepends the owner node to the target node, i.e. the owner node becomes the first child of the target node.

Parameters:

CActiveRecord	*Target node*
boolean	*Whether to validate attributes*
array	*Attributes to save*

Returns:

boolean TRUE if the node was prepended, FALSE if not

previousSibling ()

Named scope to find the previous sibling of the owner.

Returns:

CActiveRecord The owner

root ($n = 0)

Named scope to find a root node. If $n is a +ve integer the $nth root is found; anything else, the root node of the owner node is found

Parameters:

integer	*The root node number to find*

Returns:

CActiveRecord The owner

roots ()

Named scope to find root nodes.

Returns:

CActiveRecord The owner

self ()

Named scope to find the owner node. Merged using OR, so that (for example) *$model->descendants()->self()->deleteAll()* deletes the owner node and its descendants

Returns:

CActiveRecord The owner

sibling ($n = 1)

Named scope to find the nth child of the parent of the owner node.

Parameters:

integer	*The sibling number to return; $n==0 returns the last sibling. NULL will be returned if $n>lastSibling*

Returns:

CActiveRecord The owner

siblings ($*which* = self::SIBLINGS_EX)

Named scope to find the siblings of the owner. Can find all excluding (default), all including, earlier, or later siblings. By default the nodes are returned in sibling order

Parameters:

integer	*Which siblings: + self::SIBLINGS_EX - all siblings excluding the owner (default) + self::SIBLINGS_ALL - all siblings including the owner - equivalent to the parent's children + self::SIBLINGS_AFTER - later siblings + self::SIBLINGS_BEFORE - earlier siblings*

Returns:

CActiveRecord The owner

Database Schema

Below is the database schema used for unit testing. The Nested Interval Behavior requires the "nv", "dv", "snv", and "sdv" columns, though they can be renamed – see Property Details

```
CREATE TABLE `NestedInterval` (
`id`    int(11) NOT NULL AUTO_INCREMENT ,
`name`  varchar(255) NOT NULL COMMENT 'Node name' ,
`nv`    int(11) UNSIGNED NOT NULL COMMENT 'Node numerator' ,
`dv`    int(11) UNSIGNED NOT NULL COMMENT 'Node denominator' ,
`snv`    int(11) UNSIGNED NOT NULL COMMENT 'Next sibling numerator' ,
`sdv`    int(11) UNSIGNED NOT NULL COMMENT 'Next sibling denominator' ,
PRIMARY KEY (`id`)
);
```